Industry Best Practices for the Software Development Lifecycle

Quarterly Report #1
May 1, 2005 – July 1, 2005

for the
Montana Department of Transportation

By

Ray Babcock
Gary Harkin
Hunter Lloyd

Computer Science Department
Montana State University

July 14, 2005

1. Introduction

The MDT has asked us to provide industry best practices methods and tools for their software development life cycle (SDLC). We have studied the current methodology and compared it to industry practices and prepared this report and the associated documents.

An *industry best practice* is "*a technique or methodology that, through experience and research, has proven to reliably lead to a desired result."* In the area of software development, there are many different views of what constitutes a best practice. As pointed out in the call for proposals by the MDT, there are a number of organizations that offer industry best practice methodologies and they are not always compatible nor do they have a common perspective on the goals of a software development effort. Our goal is to identify a set of industry best practice techniques that fit the needs of the MDT and provide a consistent and robust process for software development.

We have looked at a wide variety of current industry methodologies and tried to identify those that are both tried-and-true, and widely accepted. We believe that the *Unified Software Development Process* [Jac1999] represents the closest thing to an industry standard for software development, and we have borrowed heavily from those methods. We also believe there is much to be learned from the Agile development methods, and we have referred to those liberally [McC1996]. The Project Management Professional (PMP) certification of the Project Management Institute (PMI) represents best practices in project management [Hel2004] and that information is used extensively in developing strategies for conducting a software project. The Capability Maturity Model Integration for Software is widely used worldwide for process improvement and reengineering, and we borrowed from those methods where appropriate [Ahe2004]. For a complete analysis of software engineering, we used *Software Engineering* [Pre2001] as a guide, but the entire area of software engineering is well developed and virtually any book will do. We have suggested the Unified Modeling Language as a representational method for developing software designs, and we recommend *UML Distilled [Boo2004]* and *UML for Mere Mortals* [Mak2005] as good user references and *The Unified Modeling Language Reference Manual* [Rum2005] for a complete analysis. In the tried-and-true category, we used information from *Joel on Software* [Spo2004], *Facts and Fallacies of Software Engineering* [Gla2003], *The Software Development Edge* [Mar2005], and *Peopleware* [DeM1999]. For user interface design, we recommend *Don't Make Me Think* [Kru2000] and *User Interface Design for Programmers* [Spo2001].

One issue that is important to discuss is how to deal with multiple viewpoints of industry *best practices*. We feel that consistency and robustness are more important to the success than having every bell-and-whistle, so we have attempted to use only a few methodologies and to combine them in a manner that provides those benefits. As mentioned previously, the *Unified Software Development Process* (USDP) [Jac1999] is the industry standard analog to the *Rational Unified Process* from Rational Systems and is widely used for software development in industry. USDP is a relatively complex process that focuses on the software development process from inception through release. While it is strong in software development it is weak in the area of project management.

The *Process Management Professional* [Hel2004] strategies have excellent recommendations for project methodologies, but do not consider software development explicitly.

The Capability Maturity Model Integration [Ahe2004] is primarily about process reengineering, which can be a part of every software development project and provides excellent practices for software project management, but the scope is too narrow to be the only source. Software CMM is the staged model from CMMI that is pertinent to software development, and it has five levels with a set of key process areas dictated for each level:

1. Initial, focus is competent people
2. Repeatable, focus is basic project management
3. Defined, focus is process standardization
4. Managed, focus is quantitative management
5. Optimizing, focus is continuous process improvement

While not specifically stated, Level 3 could be identified as the appropriate focus for this project, as the MDT attempts to standardize its software development process in accordance with industry practices. That level will be the primary focus of this work, along with some process areas from Levels 2 and 4. The key process areas are:

- *Statistical Process Management (L4)
- *Peer Reviews (L3)
- Project Interface Coordination (L3)
- *Software Product Engineering (L3)
- *Integrated Software Management (L3)
- *Organization Training Program (L3)
- *Organization Process Definition (L3)
- Organization Process Focus (L3)
- *Software Configuration Management (L2)
- *Software Quality Assurance (L2)
- Software Acquisition Management (L2)
- *Software Project Control (L2)

Those marked with an asterisk are those that are impacted by this work. Due to the limitations on the scope of this project, none of these will be completely satisfied, but all will be improved.

If you study the other *industry best practices* methodologies, you will find that they provide similar methods and characteristics. We have attempted to provide a compilation, which is at the cutting-edge technologically, but also verifiably practical and robust. We have used the existing MDT documents and templates as much as possible and have indicated changes to be made where appropriate.

1.1 Report Organization

This report is not a single document, but a series of documents that provide a strategy for implementing an industry best practices software development methodology, including templates, forms and software recommendations. The report consists of five parts:

- Recommendations and discussions of each part of the project.
- A series of processes outlining a series of steps to be followed during various phases of the project.
- Documents to be used in the various phases of the project cross-referenced in the process outlines.
- Templates and methods to be used in software used to support a project.
- Where appropriate, web pages that would be used to support a project.

The division of information that we have followed is that there is the SDLC process and several sub-processes that represent the fundamental nature of software development.

- The SDLC
- Requirements development and management
- Project management
- Software design
- Software development
- Project implementation

There is overlap between these processes in some areas, but this breakdown provides reasonably concise bodies of material with common methods that integrate into the SDLC. Each is discussed below.

2. The Software Development Lifecycle

The SDLC is the set of phases that a project must complete. The proposed phases for the MDT are Planning, System Analysis, System Design, Technical Design, Development, Implementation and Conclusion. In various references, you find different breakdowns of the SDLC, but they contain effectively the same steps. Some are software-oriented and have no planning step, and many do not differentiate between the System and Technical Design phases, and there are slight differences in the order of the steps or their content. The number of phases in the organization of the tasks is largely irrelevant within the range of those strategies that are proven to work. We believe that the MDT proposed phases are workable and meet the needs of the MDT. The use of both System and Technical Design phases is advantageous because it breaks a relatively complex phase into two parts, which can be managed more effectively.

The justification for these choices can be found in [Jac1999] which describes the USDP life cycle. It has 5 phases: Requirements, Analysis, Design, Implementation and Test. The MDT adds a planning process, which is necessary in a business context, combines

Requirements and Analysis in one phase and breaks the Design phase into two phases. That seems to be a large change, but it actually provides an identical result, pushing part of the Analysis phase into System Design and leaving the System Analysis phase as essentially the same as the USDP Requirements phase. The MDT could, theoretically, add a Requirements phase, but there seems to be no good reason for doing so. MDT process includes Test in Development and Implementation, but that is not a significant difference.

The Software CMM requirement for Organization Process Definition at Level 3 [Ahe2004] is met by this process.

Each phase of the SDLC is discussed below in terms of the justification for the tasks in the phase. There is one part of this process that is universal, and that is the *Project Planning Process,* which is a subsystem that impacts each phase. The Project Planning Process schedules and assigns resources to tasks and details the collection of data to be used in evaluating performance and improving estimates. It will be covered independently in some detail.

The SDLC process is detailed in Appendix A.

Recommended forms and templates are identified by the process from which they derive and by the SDLC phase in the form Process-Phase-Name. For example, P-P-Workscope, where P-P refers to project management and the Planning phase and the form relates to the workscope. The processes are:

- Project Management (P)
- Requirements Management (R)
- Software Analysis and Design (A)
- Software Development (S)
- Implementation (I)

And the phases are:

- Planning (P)
- System Analysis (A)
- System Design (S)
- Technical Design (D)
- Software Development (S)
- Implementation (I)
- Conclusion (C)

2.1. Planning

The planning phase is dictated by the MDT *IT Planning Process* document, which requires the preparation of an *IT Project Nomination* to be submitted for approval. The proposed Planning Phase process is designed to provide the information for the

Nomination and to establish a foundation for continuance of the project. The tasks proposed are:

- Receive the IT Nomination request.
- Develop project description.
- Identify project stakeholders.
- Develop a preliminary work scope.
- Develop an initial project team.
- Create the Use Case business model.
- Develop a preliminary deliverables manifest.
- Create a preliminary plan and schedule.
- Develop a preliminary risk assessment.
- Develop a Benefit/Cost Analysis.
- Create the IT Nomination according to stated criteria.
- If project not approved, stop.
- Plan the System Analysis phase.
- Create system analysis team list.
- Create system analysis plan.

These steps are based on information in [Ahe2004] and [Hel2004]. The CMMI specifies in Generic Process (GP) 2.2, the need to establish and maintain a plan for performing the process. The PMP mandates that project initiation and planning perform:

- An analysis of need.
- The development of project goals.
- The collection of project requirements.
- A list of project deliverables.
- The identification of project constraints.
- The development of a project schedule.
- An estimate of projected resource needs and budget.

The list of tasks above meets these requirements and honors the specifications of the MDT process. The Use Case Business Model is derived from the USDP and represents our belief that UML is the best overall design methodology for consistency throughout the lifecycle.

2.2. System Analysis

This phase takes the preliminary plan and transforms it into a detailed project plan that can be used to control and monitor the project. The list of tasks is:

- Identify project sponsor and project manager.
- Develop the requirements document.
- Develop a data dictionary.
- Develop a project work scope.

- Develop a baseline project plan.
- Finalize the project team.
- Develop the risk management plan.
- Update the project plan.
- Update benefit-cost analysis.
- Go/no-go decision.
- Plan the System Design phase.

These tasks are a combination of suggestions from the USDP and the PMP. The key deliverables from this phase are the requirements that drive the software development, and the detailed project plan that drives the lifecycle. Each of these is a dynamic document that is subject to change in succeeding phases, but those changes are an important metric to be collected and used in improving the SDLC process.

The Requirements Document and Project Plan are so important that they are treated separately in this document.

The justification for the task list can be found in the USDP [Jac1999] and in the PMP [Hel2004]. The USDP produces a Requirements Document, and the PMP suggests that the phase should produce a Workscope and Baseline Project Plan. We have added updates for the benefit-cost analysis to provide a check on the initial estimates, team selection and planning for the next phase.

A risk management plan is mandated by both the USDP and the PMP to insure that potential problems are identified and handled at each stage.

2.3. System Design

This phase converts the requirements and data dictionary into a software architecture, an implementation strategy and produces an updated project plan. The tasks are:

- Develop the system specification.
- Update requirements.
- Develop the architectural model.
- Develop the data dictionary model.
- Develop the user interface storyboards.
- Develop the documentation plan.
- Develop a user support plan.
- Develop a training plan.
- Develop a conversion plan.
- Develop a security plan.
- Develop a system test plan.
- Develop an acceptance test plan.
- Develop the acceptance test cases.

- Update the project plan.
- Plan the Technical design phase.

These tasks are derived from the USDP [Jac1999] and for the project planning portion, from the PMP. The USDP Analysis phase produces an Analysis Model, which includes a data dictionary that might be separate in a non-object-oriented programming environment. In the USDP, user interface design is in the Design phase, but we believe that best practices would break this into two parts to avoid costly mistakes. This is discussed in more detail later. The need to design Acceptance Tests and System Test early is addressed here. The project planning portion follows the requirements of the PMP [Hel2004] with regard to continuous updating of the plan. Meeting the CMMI [Ahe2004] requirements for improved quality processes adds a number of components, including a training plan, test plans, user support plans and documentation plans.

2.4. Technical Design

This phase converts the system design into a technical design that is more attuned to implementation in a programming language and produces mockups of the user interface design. The tasks are:

- Develop the functional design specification.
- Develop the data model specification.
- Develop the user interface mockups.
- Update the requirements document.
- Update the system specification document.
- Develop a unit test plan.
- Update the system test plan.
- Update the acceptance test plan.
- Create a defect tracking system.
- Update the project plan.
- Create a Development phase plan.

These tasks are derived from the USDP [Jac1999] and for the project planning portion, from the PMP. The USDP Design phase produces a Design Model, which includes a system design, subsystem designs, interface specifications and class architectures. We have added a System Specification to this list, which is a more formalized presentation of the Requirements Document. Having such a document reduces the likelihood of potential design errors. Unit tests are designed here, although the USDP suggests that this happen in the Implementation phase. We feel that performing the design here will speed implementation by not having the programming team spending its time on this activity, or possibly ignoring it. The USDP addresses the user interface design issue here as well.

The project planning portion follows the requirements of the PMP [Hel2004] with regard to continuous updating of the plan. Meeting the CMMI [Ahe2004] requirements for improved quality processes adds a defect tracking system to insure that design and

implementation errors are identified, their resolution verified and the whole process measured.

2.5. Development

This implements the system design to produce running software and documentation. The tasks are:

- Develop the running software.
- Develop the user documentation.
- Develop the product documentation.
- Develop the training courseware.
- Create a configuration management system.
- Update the requirements document.
- Update the specification document.
- Perform successful unit tests.
- Perform successful system tests.
- Update the project plan.
- Create an Implementation phase plan.

This is the Implementation phase in the USDP and fits the task list here, although it is obviously a complex activity. The PMP and CMMI best practices mandate a number of tasks based on preliminary work in earlier phases, but it also creates a configuration plan and management system as mandated by the Software CMM.

2.6. Implementation

This phase performs the installation and acceptance testing of the software and training for users. The tasks are:

- Install the beta test software.
- Collect user feedback.
- Review and update the requirements document.
- Review and update the specification document.
- Update product.
- Perform the acceptance test.
- Revise the user support plan.
- Revise the training plan.
- Perform training.
- Execute a user survey.
- Update the project plan.

This phase includes most of the Test phase of the USDP, but it includes significant tasks that attempt to achieve the process quality goals of the CMM software, including the user support plan, training and execution of a user survey. While the project is nearly

complete, it is important to perform final updates on the specification and requirements documents and the project plan to insure that any changes discovered in testing are properly accommodated and that project performance is adequately monitored.

2.7. Conclusion

This phase moves the software into the maintenance cycle, reviews the results of the implementation and the conduct of the project and makes recommendations for the future. The tasks are:

- Finalize customer approval.
- Identify and analyze problem areas in SDLC.
- Evaluate user survey results.
- Create a maintenance plan.
- Software enters the maintenance cycle.
- Review the project plan.
- Update policies and procedures

The USDP does not have this phase, but the PMP requires that a project have a closeout stage to provide feedback for further projects. Also, the CMMI model of continuous process improvement mandates that this phase continues indefinitely. It might be worthwhile to change the name of this phase to Maintenance if the MDT deems it appropriate. From a project point of view, defect tracking is an important metric to feed back into the system for purposes of better estimating project requirements and identifying design and development issues that require attention.

3. Requirements Development

The requirements are the driving mechanism for a software development project and one of the most common areas mentioned in project failures. Without well-researched and written requirements, it is impossible to meet the expectations of the users and poor requirements leave a software project with little direction and no criteria for controlling the development process. Unfortunately, good requirements are elusive because they are typically based on non-specific descriptions of needs that are voiced by people who don't understand software, and requirements creep during a project can lead to extended development time and project failure.

This is a problem of requirements development and project management. Here we discuss both, but focus on the methods needed to develop good requirements documentation.

3.1 Introduction

Even the smallest software development project benefits from clear requirements. Building a house or other engineering project without plans is inconceivable. However, many development projects begin with a brief interview and go straight to coding. The

most fundamental need according to Industry Best Practices for successful software development is to begin with a good set of requirements.

Requirements communicate "What" is to be built. They describe in detail the proposed inputs provided by the users and the proposed outputs generated by the software. Graphical user interfaces are designed based on initial figures from the requirements document.

Most people agree that the problem with developing good requirements is the problem with communication between a developer and a user. Often they don't speak the same language and failures in communication often show up in software is difficult if not impossible for normal users to use.

Many books have been written and many procedures for developing requirements have been tried by industry. These range from highly structured packages that cost many thousands of dollars down to newer ad hoc loosely structured systems such as extreme programming. The cheapest and most effective method that works for the largest variety of projects is to simply use English. Unambiguous English to be sure, but just plain pictures and plain English carefully written provide requirements that can be read and understood by both developers and users alike. They provide a written record of what the proposed software development is to create and, when carefully done, provide an unambiguous presentation. More detailed "developer oriented" software specifications can be developed using the software requirements document as a safe and secure starting point.

A good software requirements document is a dynamic, constantly changing, record of what is desired for the software to accomplish. As prototypes are shown to the users and more people read the requirements, changes can be handled by simply changing the requirements document. At some point the requirements need to be "frozen" where additional changes are held until the next version. However, error corrections and functional modifications discovered as the software development proceeds can and should still be reflected in the current requirements document. Even as the system is put into operation, the requirements document should reflect the current system. If this is done, the requirements document becomes a useful document throughout the entire life cycle of the developed software.

3.2 Requirements Elicitation

Here is where a development team shines. Elicitation of a good set of requirements is fundamental to the success of a software development project. Leffiingwell and Widrig [Lef2003] in *Managing Software Requirements* give a good list of elicitation methods.

These are:

- Using Features
- Interviewing
- Requirements Workshops
- Brainstorming and Idea Redirection
- Storyboarding

However, any sequence of activities that clearly define a set of requirements is considered useful.

3.2.1 Using Features

"System features are high-level expressions of desired system behavior"

In all the elicitation methods described, something should be written down. It does no good to the development team if one user has a clear idea of what should be done but nothing is written down to substantiate this user's vision.

High-level expressions are usually not well defined, but they can lead a development team to eliciting more specific requirements. They are often related to the user's view of system behavior. However, the development team should not leave these at a high level. They should search for the need underlying the feature described. Features and requirements, for that matter, must address real needs, not just pie-in-the-sky wishes. Another definition of features proposed by [LEF2003] is "a service the system provides to fulfill one or more stakeholder needs".

Some example features are listed below:

- Manual control of doors during fire emergency
- Provide up-to-date status of all inventoried items
- Provide trend data to assess product quality
- Report deductions-to-date by category
- Vacation settings for extended away periods
- Minimum of two independent confirmations of attack authorization required
- Windows XP compatibility

3.2.2 Interviewing

Probably the most useful and most common technique to elicit requirements is the interview. Just sitting around and talking does not work. A more structured approach is required to produce good results.

Choosing the right people to interview is important. Try to have at least one representative from each stakeholder group. These can be done at different times, but everyone needs to be included.

A good technique to use when doing an interview is the recording of a restatement of the comment provided by a stakeholder. This gives the stakeholder the ability to hear what they said repeated and correct any misconceptions on the spot. And example follows:

Developer: "How many different invoices are used in a typical month?"
User Sue: "Fred, how many would you say?"
User Fred: "Oh, I'd say probably 120?"
User Sam: "Oh that's not right, we often do over 200!"
User Sue: "How about we say 250 to be safe?"
User Fred: "Well he didn't ask for a safe number, he asked for a typical number. I'd say 175.
User Sue: "O.k. anyone have a problem with 175?
Other Users:: "no"
User Sue: "We typically use about 175 invoices each month."
Developer turns on his recorder and records "The company uses 175 invoices in a typical month."

This simple technique can provide a record of what is discussed without every cough, inane comment, joke, or other irrelevant material being recorded. It also can catch a misunderstanding if it occurs. For example suppose the following is the developer's recorded statement after the above comments:

Developer turns on his recorder and records "The company uses a maximum of 175 invoices in a typical month".
User Sue: "Wait a minute! We said that was typical not maximum"
Developer turns on his recorder and records "Correction the company uses an average of 175 different invoices in a typical month."

After the developers return to their location, the recorded sessions is played for the entire development group and a set of requirement statements are generated.

3.2.2.1 Context Free Questions

Another good technique to enhance the success of interviewing in gathering requirements is the use of "context free questions". These are questions that can be asked without regard to the context being discussed. They could be asked about ANY software development project. Using them often elicits extremely useful information to enhance the quality of the requirements gathered.

Example Context Free Questions follow: (The project name is assumed to be MDTProject1)

- Who is the client for MDTProject1?
- What is a highly successful solution really worth this client?
- Should we use a single design team, or more than one?
- How much time do we have for this project? What is the trade-off between time and value?
- Where else can the solution to this design problem be obtained? Can we copy something that already exists?
- What problems does this system solve?
- What problems could this system create?
- What environment is this system likely to encounter?
- What kind of precision is required or desired in the product?
- Am I asking you too many questions?
- Do my questions seem relevant?
- Are you the right person to answer these questions?
- In order to be sure that we understand each other, I've found that it helps to have things in writing so I can study them at leisure. May I write down your answers and give you a written copy to study and approve?
- Is there anyone else who can give me useful answers?
- Is there someplace I can go to see the environment in which this product will be used?
- Is there anything else I should be asking you?
- Is there anything you want to ask me?

Of course some of these may not be appropriate for a particular project. It will help if the entire list is scanned for each interview to be sure to include appropriate ones.

(From "Exploring Requirements Quality Before Design" by Gause & Weinberg [Gau1989].)

3.2.3 Requirements Workshops

3.2.4 Brainstorming and Idea Redirection

3.2.5 Storyboards

3.3 Stakeholders

Elicitation of a good set of requirements requires knowing all those who are involved with the project. These stakeholders can consist of state officials, any members of the development team assigned to the project, management, clerks, any person who will use this system to do productive work.

A representative of all these groups should be included in the requirements gathering process. Leaving just one important stakeholder out of the discussion can cause a failure of the system late in the development cycle when it costs a lot to fix.

3.4 Use Cases

Often the hardest part in beginning to develop a set of software requirements is simply getting started. No other way is know than to simply take the first step. A technique that is used with UML is called Use Cases. These often get the user and developer off square one and into the writing process.

A Use Case is a very simple concept. It takes an actor playing a particular role and determines "what happens" when this actor performs some activity to be controlled by the developing software system. It could be as simple as a new user "logging on" to the system or a receivables clerk handling "back orders". In any case the focus is very narrow and only the activities needed to do this particular Use Case are listed.

UML provides a diagramming technique to support Use Cases, but simple unambiguous or structured English on a yellow pad would suffice. A number of examples are provided at the end of this section.

To define a Use Case, do the following:

- Pick an actor from among the users. (system manager, payables clerk, administrator, field technician, etc.)
- Pick a roll that this actor will be performing with the software system. (logging on, entering a new vendor, accessing a payable account, adjusting an employees pay rate, etc.)
- Then, thinking about that actor performing that role, what should happen?

This will be extended with examples.

3.5 "Mary Had A Little Lamb" Heuristic

Gause & Weinberg propose a simple technique to elicit all the possibilities from a requirement statement. It is called the Mary Had A Little Lamb heuristic and is quite simple to operate. You just repeat the sentence you are analyzing putting emphasis on

each work in sequence. Then you think about the other information that might be revealed by this emphasis.

- MARY had a little lamb.
- Mary HAD a little lamb.
- Mary had A little lamb.
- Mary had a LITTLE lamb.
- Mary had a little LAMB.

Then you continue with combinations until you have all the sentence covered. Many of these words will add little or nothing to the understanding of the requirement. But, any insight gained is worth the small effort involved.

- MARY HAD a little lamb.
- Mary HAD A little lamb.
- Mary had A LITTLE lamb.
- …
- MARY HAD A LITTLE LAMB.

Try this on the requirements you are creating and watch what additional information is extracted.

The requirements process is detailed in Appendix B.

This portion is incomplete at this time.

4. Project Management

All software development life cycle methodologies suggest that there be some sort of plan for the project, but most leave the details to the interested reader. We have integrated the suggestions of the PMP [Hel2004] and other authors [*to be added*] to make the requirements for the project plan more explicit.

A project plan has nine knowledge areas [Hel2004]:

- Integration Management
- Scope Management
- Time Management
- Cost Management
- Quality Management
- Human Resource Management
- Communications Management
- Risk Management
- Procurement Management

Most of these are obvious, but Project Integration Management can be best defined as a dynamic process that coordinates all of the other parts of the plan to insure consistency. We will not discuss this process explicitly, but it will be present in the ensuing discussions.

This report will not deal with Human Resources Management, Communications or with Project Procurement Management, which are out-of-scope.

This component is needed to meet the Software CMM requirement for Software Project Control at Level 2 and Integrated Software Management at Level 3 [Ahe2004].

4.1 Scope Management

Scope Management is concerned with defining and controlling the work of the project, so it incorporates product scope and project scope. There are five processes in this activity: Initiation, Scope Planning, Scope Definition, Scope Verification and Scope Change Control.

For a software project, this reduces to translating the Requirements Document into a work plan and managing any ensuing changes. It begins in Planning phase with the development of a preliminary plan that assumes a project scope as detailed in a **scope statement**. The scope statement provides: a project justification, a product description, project deliverables, project objectives and informal project requirements. The scope statement if refined in the System Analysis phase and is used to create the **Work Breakdown Structure** (WBS) which maps out the project deliverables and reduces each to a set of identifiable tasks.

The Requirements Documents is superceded by the System Specification which may result in some updates to the WBS, and these changes will continue into the maintenance cycle. The Requirements Document, System Specification and WBS are living documents that will change continuously during the project and must at all times, reflect the current understanding of the project team regarding product requirements and task assignments.

4.2 Time Management

Time Management has five processes: Activity Definition, Activity Sequencing, Activity Duration Estimation, Schedule Development and Schedule Control. The Activity Definition begins with the WBS and Scope statement, and in conjunction with historical data, constraints and assumptions about the project, is used to reduce the WBS to a **Task List** of activities that are identifiable and assignable. There is no specific criteria for defining a task, but it should be small enough that the time estimate of the complete is accurate to within some acceptable margin of error. Early in the process, it might be reasonable to estimate development time as 3 months, assuming a possible error of 1 month, while later individual development tasks might be required to be estimated within 1 day. The Task List will be under constant change as the design and development continue. The Task List should contain fields for the following:

- The task identifier
- The task description
- The estimated duration, minimum, expected and maximum
- Resources required
- Estimated resource cost
- Estimated total cost
- Actual time required
- Actual cost
- Notes from the project manager

The second time management structure is the **Precedence Diagram** which shows the dependency relationship between the tasks. This document can show the project manager the appropriate assignments to make to avoid bottlenecks and stoppages. It is based on the time estimates in the task list and generates a set of paths through the project to completion. It is also common to perform a **Critical Path** analysis of the precedence diagram to determine which activities constitute the longest path through the task list, which is called the critical path.

As with the scope, the time management tools are dynamic and must be updated continuously to reflect new information and to collect data.

4.3 Cost Management

Cost Management has four processes: Resource Planning, Cost Estimating, Cost Budgeting and Cost Control. Cost estimating produces data into the Task List and Precedence Diagram so there are no new documents, but successful project management depends on the collection and use of data for future estimates. Cost estimates are needed for the **Benefit/Cost Analysis** and for improving project cost estimates**.**

4.4 Quality Management

Quality Management has three processes: Quality Planning, Quality Assurance and Quality Control. Quality in software can be objective or subjective. Objective measures are surveys of user satisfaction, lower operating costs, higher productivity, lower defect and rework rates. It is important to collect as much information as possible, but the reality is that information outside of the development group is expensive to collect and is typically not done.

Most of these issues are outside of the scope of this document, but a **defect tracking system** is part of the process, as is user feedback on software, training and documentation.

4.5 Risk Management

Risk Management has six processes: Risk Management Planning, Qualitative Risk Analysis, Quantitative Risk Analysis, Risk Response Planning, and Risk Monitoring and Control. Risk can be defined as potential costs in time or other resources that could be incurred. This impacts the SDLC is two ways: there could be risk associated with

estimates of time or resources in the development process, and there could be risks associated with factors outside of the SDLC such as changing needs. External risks could be due to:

- Budgetary changes
- Political issues
- Legal issues
- Environmental issues

The risks in the SDLC could be due to:

- Schedule slip
- Scope slip
- Technical issues
- Personnel issues

Each of these presents some danger that the project will fail or will incur more costs than expected, possibly more than is tolerable. While these risks can only be partially controlled, they can be planned for. During the planning stage, a **Risk Assessment** will identify potential risks to the project, evaluate their potential impact and determine under what conditions a project might be halted or reviewed. At this stage, it might be necessary to halt a project if the risks are substantial.

During the System Analysis phase, a **Risk Management Plan** for the SDLC will be developed which will detail what risks are to be considered and how their impact should be evaluated with regard to the project. A **Risk Assessment** document prepared at each stage is used to evaluate the risk under the Risk Management Plan.

For example, during the System Design phase, the project team might determine that a particular task involves technology for which they lack experience. This is a higher than normal risk, because the development time estimates, and possibly even the feasibility of the project, are in question. The team and project manager should evaluate this potential risk in determining how to proceed. One possible risk reduction strategy borrowed from Agile development is to have part of the team pursue this development until they are certain of the time and cost estimates and then use the new data.

The project management process is detailed in Appendix C.

5. Software Design

This component of the project comes exclusively from the USDP and is concerned with developing an Architectural Model and Design Model primarily on the Requirements Document. We are proposing to use UML for these processes as is used in the USDP and documented in [Boo2004], [Mak2005] and [Rum2005]. The methods described here meet the Software CMM requirements for Software Product Engineering at Level 3.

These methods will detailed in the next report.

6. Software Development

This component of the project is described in broad terms by the USDP and is concerned with converting the Design Architecture into running software. This phase is difficult to describe in detail because it depends significantly on the languages used, the target software and hardware platforms and the underlying goals, but there are standard methods for managing this process that can be proposed. The methods described here meet the Software CMM requirements for Peer Reviews at Level 3 and Software Quality Assurance at Level 2.

These methods will detailed in the next report.

7. Project Implementation

This component is not specified by any standard process, but represents a collection of activities that are vital to long-term success and quality control. The Software CMM requirements for an Organization Training Program at Level 3, Software Configuration Management at Level 2.

These methods will detailed in the next report.

Appendix A

Software Development LifeCycle Process

Use

Implement the software development life cycle.

Deliverables

- Installed software
- User Training
- User documentation
- Customer support plan

Personnel

- Project sponsor
- Project manager
- Software development team
- Trainers
- Technical writers

Planning

Starting Condition:

IT nomination request received or expected.

Deliverables:

Preliminary project description
Preliminary project plan and budget
Benefit/Cost Analysis
IT Nomination packet

Tasks:

Requirements development and management

Develop project description (Form ProjDesc).
Identify project stakeholders.
Develop a preliminary deliverables manifest (Form DelivManifest).

Project management

Develop an initial project team (Form TeamManifest).
Develop a preliminary work scope (Form WorkScope).
Create a preliminary plan and schedule (Process ProjectManage).
Develop a preliminary risk assessment (Form RiskAssess).
Develop a Benefit/Cost Analysis (Form BCAnalysis).
Plan the System Analysis phase (Process ProjectManage).

Software design

Create the Use Case business model (Form BusModel).

Software development

Project implementation

System Analysis

Starting Condition:

Decision is made to proceed with the project.

Deliverables:

Requirements document
Data dictionary
Project workscope
Baseline project timeline
Baseline project task list
Risk management plan
Updated Benefit/Cost Analysis

Tasks:

Requirements development and management

Develop the requirements document (Process DevelopRD).

Project management

Identify project sponsor and project manager (Form InitiateProject).
Develop a project work scope (Form WorkScope).
Develop a baseline project plan (Process ProjectManage).
Develop the risk management plan (Form RiskTgmtPlan).
Finalize the project team (Form TeamManifest).
Update benefit-cost analysis (Form BCAnalysis).
Plan the System Design phase (Process ProjectManage).

Software design

Develop a data dictionary (Process DevelopRD).

Software development

Project implementation

System Design

Starting Condition:

Requirements document has reached a stable state.

Deliverables:

System specification
Updated project plan
Software Analysis Model
User interface storyboards
Security plan
Acceptance test plan and test cases
Documentation plan
User support plan
Training plan
Conversion plan

Tasks:

Requirements development and management

Develop the system specification (Form SysSpec).

Update requirements (Form RD).

Project management

Update the project plan (Process ProjectManage).
Plan the Technical design phase (Process ProjectManage).

Software design

Develop the Analysis Model (Process AnalysisModel).
Develop the data dictionary model (Form DataDictModel).
Develop the user interface storyboards (Form UserSB).

Software development

Project implementation

Develop a security plan (Form SecPlan).
Develop a system test plan (Form SysTestPlan).
Develop an acceptance test plan (Form AccTestPlan).
Develop the acceptance test cases (Form AccTestCase).
Develop the documentation plan (Form DocPlan).
Develop a user support plan (Form CustSuppPlan).
Develop a training plan (Form TrainingPlan).
Develop a conversion plan (Form ConversionPlan).

Technical Design

Starting Condition:

IT nomination request received or expected.

Deliverables:

Software Design Model
User interface mock-ups
Unit test plan
Update system specification and requirements document
Defect tracking system
Updated project plan

Tasks:

Requirements development and management

Update the requirements document (Form RD).
Update the system specification document (Form SysSpec).

Project management

Update the project plan (Process ProjectManage).
Create a Development phase plan (Process ProjectManage).

Software design

Develop the Design Model (Process DesignModel).
Develop the data model specification (Process DataModelDesign).
Develop the user interface mockups (Form UserIntMockup).

Software development

Develop a unit test plan (Form UnitTestPlan).

Project implementation

Create a defect tracking system (Form DefectTrack).
Update the system test plan (Form SysTestPlan).
Update the acceptance test plan (Form AccTestPlan).

Development

Starting Condition:

IT nomination request received or expected.

Deliverables:

Functioning and tested software system.
User documentation
Product documentation
Training courseware
Configuration management system

Tasks:

Requirements development and management

Update the requirements document (Form RD).
Update the specification document (Form SysSpec).

Project management

Update the project plan (Process ProjectManage).
Create an Implementation phase plan (Process ProjectManage)

Software design

Update the system design

Software development

Develop the running software (Process Develop).
Perform successful unit tests (Form UnitTestCert).

Project implementation

Develop the user documentation (Form UserDoc).
Develop the product documentation (Form ProductDoc).
Develop the training courseware (Form TrainingDoc).
Create a configuration management system (Form ConfigMgmt).
Perform successful system tests (Form SysTestCert).

Implementation

Starting Condition:

IT nomination request received or expected.

Deliverables:

Training surveys
User response surveys
Completed acceptance test
Updated and tested software
Updated user support plan

Tasks:

Requirements development and management

Review and update the requirements document (RD).
Review and update the specification document (SysSpec).

Project management

Update the project plan (Process ProjectManage).

Software design
Software development
Update product (Process ImplementUpdate).
Project implementation
Install the beta test software (Form BetaInstall).
Collect user feedback (Form UserFeedback).
Perform the acceptance test (Form AccTestCert).
Revise the user support plan (Form CustSuppPlan).
Complete training (Form TrainPlan).
Perform training (Process Train).
Execute a user survey (Form UserSatisfaction).

Conclusion
Starting Condition:
IT nomination request received or expected.
Deliverables:
Customer approval
Software in maintenance cycle
Analysis of project performance
Tasks:
Requirements development and management
Project management
Create a maintenance plan (Form MaintenancePlan).
Review the project (Process ProjectManage).
Update policies and procedures.

Software design
Software development
Project implementation
Finalize customer approval (Form FinalApproval).
Evaluate user survey results (Form UserSurveyResponse).
Software enters the maintenance cycle (Process MaintCycle).

Appendix B
Requirements Development

Appendix C
Project Management
with Microsoft Project 2003

Introduction

It is the desire of the MDT to use Microsoft Project 2003 to support project management, and we believe that is an reasonable choice. In the following, we will discuss the various tasks and deliverables expected from the project management system and how they can be managed with Project 2003. It is not productive to separate the Project 2003 portion of project management from any other parts, so the entire project will be discussed here. Templates and forms will be delivered in electronic format at the completion of the project.

The project management system is responsible for the following deliverables:

- Project schedules
- Work assignments
- Cost and time estimates
- Resource assignments
- Project assessments

These will be discussed in the context of the SDLC process described earlier.

Planning

The planning phase requires some preliminary project management efforts in order to develop cost estimates and to insure that resources are available for a project. Typically, the minimal effort is desirable because the project is not yet approved or funded, but accurate estimates are important to avoid over-committing resources or discouraging work. The deliverables are:

- Preliminary workscope
- Preliminary project team
- Preliminary schedule
- Preliminary risk assessment
- Benefit/Cost analysis
- Plan for system analysis phase (if project to proceed)

Preliminary Workscope

The workscope details the work that is expected on the project. At this point, the details remain sketchy, but the document must precisely specify an understanding between the two parties as to what is expected. A workscope document must contain:

- A list of deliverables.
- What work is to be accomplished.

- Specific exclusions to the scope of work.
- A list of milestones marking project progress.
- How changes to scope will be handled.

The MDT workscope form requires a *Business Processes* entry to indicate expected changes in business processes. Another item that is not typically mandated by any best practice but could be useful is any interaction between this system and any other software systems.

This document is identified as **Form P-P-Workscope** and requires the project manager to convert the project description, assumptions and constraints into a list of project deliverables, goals and justification. This document is the precursor to the Requirements Document which will drive the development process and should be based on the best information available regarding the desired outcomes of the project. The workscope at this point is still preliminary and won't be finalized until the next phase.

Preliminary Project Team
This document is identified as **Form P-P-Team** and should specify the expertise required, not individuals. This information will be moved into Project 2003 in order to provide a basis for narrowing the assignments in the next phase and to provide historical information for assessing the accuracy of the estimate.

Preliminary schedule
The preliminary schedule should attempt to identify the expected starting and completion dates for each phase, and of course, the total project time. At this stage, these estimates are subject to significant error, but a reasonable estimate of cost depends on this data. This information will be entered directly into Project 2003 using **Template P-P-PrelimSched**.

Preliminary Risk Assessment
The risk assessment is preliminary at this stage, and should include potential sources of risk in the project and the risk potential. At this stage, an identifiable large risk could result in the project being terminated, such as the possibility that users will resist conversion or that personnel vacancies will delay the project unacceptably. **Form P-P-Risk** will be used for this task. One risk is that better analysis will indicate that the project costs are likely to be much higher than anticipated.

Benefit/Cost analysis
A benefit/cost analysis attempts to determine if the balance between the benefits and cost of a project justify proceeding. The difficulty is that the benefits are often not monetary, so a benefit could require some judgment as to its value, or it could be a requirement that mandates the project be completed. Nevertheless, this process, as implemented by **Form P-P-BCAnalysis** is important and should not be ignored.

Plan for System Analysis Phase
In order to proceed to the next phase, the project team should determine who will provide which services and what the expected timeline should be. There is a bit of overlap with the next phase, as the most likely scenario is that the project management deliverables from that phase will be used to make this determination. However, the personnel involved in the project need to be aware of their roles and their must be a timeline specifying when the phase is to be complete and any milestones that are required.

System Analysis

In this phase, the principals on each side of the project (customer and development) need to be identified and the preliminary plan needs to expanded to as great a degree as possible at this time. This is also the time to make certain decisions about the conduct of the project, such as risk management, and to update the benefit/cost analysis to insure that the project is still tenable.

The deliverables are:

Project management

Identify project sponsor and project manager.
Develop a project work scope.
Develop a baseline project plan.
Finalize the project team.
Update benefit-cost analysis.
Develop the risk management plan.
Plan the System Design phase.

Project Principals
The project principals are the project manager and the project sponsor (on the customer side). These two people are responsible for the communication and integration between the two parties and their roles must be established early in the process. Other stakeholders should also be identified to as great an extent as possible. For example, if the sponsor elects people to represent him in the requirements solicitation process or in designing training protocols. The form for this activity is **P-A-Principals**.

Workscope
The workscope developed in the Planning phase should be updated and finalized to as great an extent as possible. The project manager and sponsor should execute a formal agreement regarding the content of this document.

Baseline Plan

The workscope and preliminary plan are used to create a baseline plan, consisting of a task list and timeline. These will be developed using Microsoft Project 2003 as **Tmpl-P-A-Tasklist** and **Tmpl-P-A-Timeline.**

Project Team
Given the task list and the workscope, it should be possible to finalize the project team members, what role they will play in the project and the expected demands on their time. Assignments and tracking of effort will be managed using MP2003.

Updated Benefit-Cost Analysis
At this point, the benefit-cost analysis should be updated showing the new estimate of costs, and a decision should be made as to how to proceed. This process should be repeated each time the plan is updated significantly, but this is the first time where considerable effort has gone into estimating staffing requirements, so it is an opportunity to test the accuracy of earlier estimates and to insure that costs are not increasing significantly.

Risk Management Plan
The risk management plan details what risks are considered to be potential threats to the timely completion of the project, how they should be managed and if there are points in time where the project should be reviewed before continuing. **Form P-A-RiskManPlan** collects some data, but many aspects of this activity are subjective.

More to come here

System Design Phase Plan
The plan for completing the System Design phase should be completed here. All personnel involved should be accounted for and their time made available, all resources should be in place, and specific procedures and policies covering this part of the project should be known by all involved. This includes any software to be used for the design, databases or repositories needed, methods to be used in the design and standards for documenting the design.

System Design
In this phase, the plan needs to be updated to reflect any changes and to plan the next phase. The deliverables are:

- Update the project plan.
- Plan the Technical design phase.

Plan Update
Technical Design Plan

Technical Design

In this phase, the plan needs to be updated to reflect any changes and to plan the next phase. The deliverables are:

- Update the project plan.
- Create a Development phase plan.

Plan Update
Development Plan

The deliverables are:

Development
In this phase, the plan needs to be updated to reflect any changes and to plan the next phase. The deliverables are:

- Update the project plan.
- Create an Implementation phase plan.

Plan Update
Development Plan

Implementation
In this phase, the plan needs to be updated to reflect any changes and to plan the next phase. The deliverables are:

- Update the project plan.

Plan Update

Conclusion
In this phase, there is actually considerable planning effort. The maintenance cycle begins and that requires planning, and this is the point in time where the project is reviewed, the performance of the plan is judged and significant amounts of measurement data are processed and used to update policies and procedures.

The deliverables are:

- Create a maintenance plan.
- Review the project.
- Update policies and procedures.

Create a Maintenance Plan
Review the project
Update Policies and Procedures

Appendix D
Software Design Details

Objective:

Manage the software development lifecycle and collect data to be used for process improvement and reengineering.

Deliverables

- Analysis model
- Design model

Planning
System Analysis
System Design
Technical Design
Development
Implementation
Conclusion

Appendix E
Software Development Process

Use

Manage the software development lifecycle and collect data to be used for process improvement and reengineering.

Deliverables

- Tested software
- User Documentation
- Product documentation

Personnel

- Project manager
- Software development team

Planning
System Analysis
System Design
Technical Design
Development
Implementation
Conclusion

Appendix F
Forms

Our Form	MDT Form
P-P-BCAnalysis	
P-P-Prelim	
P-P-Risk	
P-P-Team	
P-A-Principals	
P-A-RiskManPlan	
P-P-Workscope	PMM05_SCOPE_STATEMENT_01.DOC

Appendix G
Templates

Our Form	MDT Form
P-P-PrelimSched	
P-A-Tasklist	
P-A-Timeline	

References

[Ahe2004] Ahern, D., et al, *CMMI Distilled,* Addison-Wesley, 2004.

[Bro1995] Brooks, F., *The Mythical Man-Month,* Addison-Wesley, 1995.

[DeM1999] DeMarco, T., Lister, T., *Peopleware*, Dorsett House Publishing, 1999.

[Fow2004] Fowler, M., *UML Distilled, Third Edition, Addison-Wesley Pearson,* 2004.

[Gau1989] Gause, D., Weinberg, M., Exploring Requirments: *Quality Before Design, Dorsett House Publishing,* 1989.

[Gla2003] Glass, R., *Facts and Fallacies of Software Engineering,* Addison-Wesley Pearson, 2003.

[Hel2004] Heldman, *Project Management Professional Study Guide, 2nd Edition*, Sybex, 2004.

[Jac1999] Jacobsen, I., et al, *The Unified Software Development Process*, Addison-Wesley, 1999.

[Lef2003] Leffingwell, D, Widrig, D., M*anaging Software Requirments:A Use Case Approach, Second Edition*, Addison-Wesley, 2003.

[Kru2000] Krug, S., *Don't Make Me Think*, New Riders Press, 2000.

[Mak2005] Maksimchuk, R., Naiburg, E., *UML for Mere Mortals*, Addison-Wesley Pearson, 2005.

[Mar2005] Marasco, J., *The Software Development Edge*, Addison-Wesley Pearson, 2005.

[McC1996] McConnell, S., *Rapid Development*, Microsoft Press, 1996.

[Mel2004] Mellor, S., el al, *MDA Distilled: Principles of Model-Driven Architecture,* Addison-Wesley, 2004.

[Pre2001] Pressman, R., *Software Engineering,* McGraw, 2001.

[Rum2005] Rumbaugh, J., et al, *The Unified Modeling Language Reference Manual,* Addison-Wesley, 2005.

[Spo2001] Spolsky, J., *User Interface Design for Programmers*, Apress, 2001.

[Spo2004] Spolsky, J., *Joel on Software*, Apress, 2004.